CO 351 Network Flow Theory

Jude Gao
University of Waterloo

2019 Spring

Contents

Lecture 1

Feasibility Characterization of MCFP

The auxiliary problem for MCFP is similar to the auxiliary TP. Thus, let's recall the auxiliary TP.

Definition 1.1 (Auxiliary TP)**.** Given a TP with digraph $D = (N, A)$ and node demands $b \in \mathbb{R}^N$, the auxiliary TP with $D' = (N', A')$, node demands $b' \in \mathbb{R}^N$, arc costs $w' \in \mathbb{R}^A$ is:

- $N' = N \cup \{z\}$ (z is a new node)
- $A' = A \cup \{vz : v \in N, b(v) < 0\} \cup \{zv : v \in N, b(v) > 0\}$
- $b'(z) = 0$, and $b'(v) = b(v)$ for $v \in N$
- $w'(e) = \begin{cases} 0 & \text{if } e \in A \text{ (original arcs)} \\ 1 & \text{if } e \in A' \setminus A \text{ (new arcs)} \end{cases}$

Let's come back to MCFP. The result is quite obvious but still worth mentioning:

Theorem 1.2. *A MCFP is feasible if and only if its corresponding auxiliary LP has optimal value* 0*.*

We will now try to characterize the feasibility of the auxiliary MCFP. Our goal is to find a certificate of infeasibility, so let's start off by assuming that our MCFP is infeasible. Then, by Theorem 1.2, we know that the auxiliary MCFP has a positive optimal value. We wil further assume that the optimal flow in auxiliary MCFP is a tree flow.

Set potentials for y so that $y_z = 0$. Then, all other nodes have potentials either 1 or -1. Moreover, we know that the nodes with positive demands have potential 1, and the nodes with negative demands (i.e., supply nodes) have potential -1.

Let S_1 be a set of all the nodes with potential 1. Note that these nodes have positive demands. Likewise, let S_{-1} be a set of all the nodes with potential -1.

Consider an arc e from S_{-1} to S_1. $\overline{w_e} = -1 + 0 - 1 < 0$, so e must be at capacity. Consider an arc e from S_1 to S_{-1}. $\overline{w_e} = 1 + 0 + 1 > 0$, so e must be at zero. Since the tree flow in the auxiliary MCFP is a feasible flow, then the net flow of S_1 equals the sum of demands in S_1, that is

$$x(\delta(\overline{S_1})) - x(\delta(S_1)) = b(S_1)$$

Since the flow of all outward arcs is at zero, we obtain

$$x(\delta(\overline{S_1})) = b(S_1)$$

Since the flow of all inward arcs is at capacity, and the flow from z is still positive, we know that the net flow in S_1 is greater than the sum of capacities of inward arcs, i.e.

$$x(\delta(\overline{S_1})) = b(S_1) > c(\delta(\overline{S_1}))$$

Then, S_1 is a subset of nodes with the property that $b(S_1) > c(\delta(\overline{S_1}))$. This case is obtained in the following theorem:

Theorem 1.3 (Feasibility Characterization Theorem)**.** *A MCFP is infeasible if and only if there exists $S \subseteq N$ such that $b(S) > c(\delta(\overline{S}))$.*

Note that the above sentences have argued the existence of such set implies infeasibility. While it is hard to prove, it is indeed the case that every infeasible auxiliary MCFP admits such a set.

Lecture 2

Shortest Dipaths Problem

2.1. TP Formulation

Given a digraph $D = (N, A)$, arc costs $w \in \mathbb{R}^A$, and two distinct nodes $s, t \in N$, the goal is to find a shortest s, t-dipath.

Our first attempt is to formulate this problem into TP. To do that, we will use the same D and w, but we will introduce the node demands b such that $b_s = -1$, $b_t = 1$, and $b_v = 0 \, \forall v \notin \{s, t\}$.

We have specified b, w for this TP formulation. What remains is to think about the objective function and the constraints. Recall that the objective function for TP is simply $w^T x$, which remains true in this case. Also recall that the constraints for LP are $x(\delta(\bar{v})) - x(\delta(v)) = b_v$; in other words, the netflow should be equal to the node demand for every node. Here, we know that the values for b have only 3 possibilities, namely, -1, 1, and 0, and we can say more about where these values can appear:

- If $v = s$, the netflow at v should be -1.
- If $v = t$, the netflow at v should be 1.
- If $v \notin \{s, t\}$, the netflow at v should be 0.

Hence, we may define the constraints for this TP more precisely as follows

$$x(\delta(\bar{v})) - x(\delta(v)) = \begin{cases} -1 & \text{if } v = s \\ 1 & \text{if } v = t \\ 0 & \text{otherwise} \end{cases}$$

for every node $v \in N$.

In summary, we formulated the shortest dipaths problem into the TP of the following form:

$$\begin{aligned}
&\min \quad w^T x \\
&\text{s.t.} \\
&x(\delta(\bar{v})) - x(\delta(v)) = \begin{cases} -1 & \text{if } v = s \\ 1 & \text{if } v = t \\ 0 & \text{otherwise} \end{cases} \\
&x \geq 0
\end{aligned}$$

Now, the dual problem for this TP formulation is revealed:

$$\begin{aligned}
&\max \quad y_t - y_s \\
&\text{s.t.} \\
&y_v - y_u \leq w_{uv} \quad \forall uv \in A
\end{aligned}$$

Note that previously we have seen the dual objective $b^T y$ for the standard dual TP. You should not be surprised to learn that $b^T y$ and $y_t - y_s$ are equivalent in this TP formulation. That is because $b_v = 0$ for any node $v \neq s$ or $v \neq t$; moreover, $b_s = -1$ and $b_t = 1$, which is boiled down to $y_t - y_s$. The constraints are exactly the same as the standard dual TP constraints: the reduced cost must be non-negative.

We are likely to think about the CS conditions:

$$x_u v \neq 0 \implies \overline{w_{uv}} = 0 \quad \forall uv \in A$$

However, we will quickly realize that the optimal solution to this TP formulation may not give a s,t-dipath at all. What to do if the optimal solution is a cycle remains unsolved.

Before considering that, we will have a definition that gives a way of thinking dipaths as feasible flows to the TP formulation:

Definition 2.1. Let P be an s,t-dipath. The characteristic vector of P is a flow $x^P \in \mathbb{R}^A$ such that

$$x_e^P = \begin{cases} 1 & \text{if } e \in A(P) \\ 0 & \text{if } e \notin A(P) \end{cases}$$

It is easy to see that the characteristic vector of an s,t-dipath is feasible to our TP formulation.

2.2. Flow Decomposition Theorem

Now, it is time to think about the possibility that the optimal solution to this TP formulation is not an s,t-dipath. We will see that there is not much to worry about from the following theorem:

Theorem 2.2 (Flow Decomposition Theorem)**.** *If $\bar{x}$ is an integral feasible solution to our LP, then $\bar{x}$ is a sum of the characteristic vectors of an s,t-dipath and a collection of dicycles.*

Proof. Let $F = \{e \in A\overline{x_e} > 0\}$ denote a set of flow that is positive. Let $\delta(S)$ be any s,t-cut. The netflow is -1, so there must be at least one arc in $\delta(S)$ with nonzero flow. Then, there exists an s,t-dipath P. Let $x' = \bar{x} - x^P$. Since both $\bar{x}$ and x^P satisfy the flow constraints,

$$x'(\delta(\bar{v})) - x'(\delta(v)) = 0$$

for all $v \in N$. □

Lecture 3

Rooted Trees and Ford's Algorithm

Last time we introduced the single-source shortest path problem. We are looking for a spanning tree rooted at s representing shortest dipaths. We will continue studying some properties of rooted trees, which you will also see on the assignments.

3.1. Rooted Trees

Proposition 3.1. *Let $D = (N, A)$, $s \in N$.*

1. *There is an s,t-dipath in D for each $t \in N$ if and only if there exists a spanning tree in D rooted at s.*
2. *Let T be a spanning tree in D. Then T is rooted at s if and only if the in-degree of s is 0 and the in-degree of all other nodes is 1 in T.*

Definition 3.2 (Predecessor)**.** In a spanning tree rooted at s, for each node v, its predecessor is the unique node u such that uv is an arc in T.

These are enough for us to talk about an algorithm for the shortest dipath problem.

3.2. Ford's Algorithm

Main Idea

Solve all s,t-dipath problems for a fixed s. It produces a spanning tree of shortest paths rooted at s, and a feasible potential where all tree arcs are equality arcs. We can produce the tree by keeping track of the predecessor for each node.

Algorithm in Simple Form

Initialize potentials y, predecessor p.
While y is not feasible, find a bad arc uv, correct uv.

Full Algorithm

1. Initialization: Set $y_s = 0$, $y_v = \infty$ for $v \in N \setminus \{s\}$. Set predecessor $p_v =$ `undefined` for $v \in N$.
2. While y is not feasible...
   a) Find an arc $uv \in A$ where $\overline{w_{uv}} < 0$, i.e., $y_u + w_{uv} < y_v$.
   b) Set $y_v = y_u + w_{uv}$, set $p_v = u$.

Remark 3.3. y_v never increases: by setting $y_v = y_u + w_{uv}$ where $y_v > y_u + w_{uv}$ to begin with, we decrease y_v.

Definition 3.4 (Predecessor Digraph)**.** At any time of the algorithm, the predecessor digraph D_p is one where $N(D_p) = N$, and $A(D_p) = \{p_v v : v \in N \setminus \{s\}\}$.

Proposition 3.5. *Throughout the algorithm, $\overline{w_e} \leq 0$ for all arcs in D_p.*

Proof. Focus on a node v and consider the arc $p_v v$. When we correct $p_v v$, its reduced cost $\overline{w_{p_v v}} = 0$. y_v does not change unless we change its predecessor through a correction, so $\overline{w_{p_v v}} = 0$. If y_{p_v} changes (due to correcting other arcs), then by Remark 3.3, y_{p_v} is reduced. Thus, the reduced cost of $p_v v$ is also reduced, so $\overline{w_{p_v v}} < 0$. □

Lecture 4

Bellman-Ford Algorithm

4.1. Ford's Algorithm Cont'd

We will finalize Ford's Algorithm and introduce Bellman-Ford Algorithm.

Proposition 4.1. *If D_p contains a dicycle, then D contains a negative dicycle, and the algorithm does not terminate.*

Proof. Suppose we produce a dicycle C in D_p by correcting the arc uv. Say C is $v = v_1, v_2, \ldots, v_k = u, v_1$. Since we corrected uv, $y_u + w_{uv} - y_v < 0$. This means $y_{v_k} + w_{v_k v_1} - y_{v_1} < 0$. By Proposition 3.5, $w_e \leq 0$ for all arcs in D_p. Then, $y_{v_i} + w_{v_i v_{i+1}} - y_{v_{i+1}} \leq 0$ for $c = 1, \ldots, k-1$. Adding all inequalities, y's cancel out, so $w_{v_1 v_2} + \cdots + w_{v_k v_1} < 0$, so C is a negative dicycle.

From assignment, if a negative dicycle exists, then no feasible potentials exist, so the algorithm does not terminate. □

Proposition 4.2. *If the algorithm terminates, then D_p is a spanning tree of shortest dipaths rooted at s with y_v as the cost of a shortest s, v-dipath.*

Proof. Since our algorithm terminates, D_p cannot contain a cycle. Thus, D_p is a spanning tree. Since all nodes other than s have a predecessor, the in-degree is 1, and the in-degree of s is 0. Then, D_p is rooted at s.

Now all arcs in D_p are equality arcs since $\overline{w_e} \leq 0$ for all $e \in A(D_p)$, and $\overline{w_e} < 0$ is impossible as y is feasible; otherwise, the algorithm would have not been terminated. For $v \in N$, consider the LP formulation for shortest s, v-dipath. Let P be the s, v-dipath in D_p. Then, x^P is feasible for the LP, and y is feasible for the dual. CS conditions hold since all arcs in P are equality arcs. By duality, they must be optimal. The optimal value is the cost of P. The objective of the dual is $y_v - y_s = y_v - 0 = y_v$, so y_v is the cost of a shortest s, v-dipath. □

4.2. Bellman-Ford Algorithm

Bellman-Ford Algorithm is a refinement of Ford's Algorithm. The idea of the algorithm is to go through arcs in "passes" and do at most $|N| - 1$ arcs.

Algorithm

1. Initialization: same as Ford's
2. Set $i = 0$
3. While $i < |N| - 1$ and y is not feasible...
   a) $i = i + 1$
   b) For each $uv \in A$, if $\overline{w_{uv}} < 0$, then set $y_v = y_u + w_{uv}$, set $p_v = u$.

Two Types of Termination

1. If a feasible potential is found, everything from Ford's Algorithm applies, and we have an optimal tree.
2. If $i = |N|$, then we must have a negative dicycle.

Runtime Analysis

The algorithm runs at most $|N| \cdot |A|$ steps.

Correctness

The main idea of why $|N| - 1$ iterations: Imagine a shortest s, v-dipath. It uses at most $|N| - 1$ arcs.

- In the first pass, we must have at least corrected the first arc on the dipath.
- In the second pass, we must have at least corrected the second arc on the dipath.
- and so on...

Hence, the s, v-dipath must be corrected in at most $|N| - 1$ arcs.

Lecture 5

Bellman-Ford Algorithm and Dijkstra's Algorithm

We sketched the proof of the correctness of Bellman-Ford Algorithm last time. We will prove it formally here. At the end, we will start a new topic – Dijkstra's Algorithm.

5.1. Bellman-Ford Algorithm Cont'd

Remark 5.1. For Bellman-Ford Algorithm, the first thing we need to do is to agree on an order of arcs. For every iteration, we check and possibly correct arcs in the order.

The remainder of the discussion of Bellman-Ford is its correctness. To prove Theorem 5.3, we need the following proposition:

Proposition 5.2. *Suppose no negative dicycle exists. Then at any point in the algorithm,* $y_v \geq d_v$.

Proof. It is obviously true at initialization. If $y_v \neq \infty$, then we can trace v back to s using D_p, since no negative dicycle exists. We know $\overline{w_e} \leq 0$, so $y_v \geq w(P)$, where P is the s, v-dipath. In particular, we know $y_v \geq w(P) \geq d_v$. □

Theorem 5.3. *Let* d_v *be the cost of a shortest* s, v*-dipath. Suppose no negative directed cycle exists. After the* i*-th iteration, if there is a shortest* s, v*-dipath using at most* i *arcs, then* $y_v = d_v$.

Proof. We prove by induction on i. It is trivial for $i = 0$. Assume true after i-th iteration. Show true after $(i+1)$-th iteration. If there is a shortest s, v-dipath using at most i arcs, by induction, $y_v = d_v$. By Proposition 5.2, y_v will not go below d_v. Suppose now there is a shortest s, v-dipath P that uses $i+1$ arcs, say $s =$

$v_0, v_1, \ldots, v_{i+1} = v$. Since no negative dicycle exists, $v_0, \ldots, v_i$ is a shortest v_0, v_i-dipath using i arcs. By induction, $y_{v_i} = d_{v_i}$, and this remains true in the $(i+1)$-th iteration. Consider $\overline{w_{v_i v_{i+1}}}$. If $\overline{w_{v_i v_{i+1}}} = 0$, then $y_{v_{i+1}} = d_{v_i} + w_{v_i v_{i+1}} = w(P) = d_{v_{i+1}}$. If $\overline{w_{v_i v_{i+1}}} > 0$, then $y_{v_{i+1}} < d_{v_i} + w_{v_i v_{i+1}} = d_{v_{i+1}}$, contradicting Proposition 5.2. If $\overline{w_{v_i v_{i+1}}} < 0$, the algorithm will correct in the $(i+1)$-th iteration. Then, $y_{v_{i+1}}$ is set to $y_{v_i} + w_{v_i v_{i+1}} = d_{v_{i+1}}$. □

Corollary 5.4. *At the end of Bellman-Ford, if y is feasible, then $y_v = d_v$ for all $v \in N$. Otherwise, there exists a negative dicycle.*

Proof. Bellman-Ford runs $|N| - 1$ iterations. Any shortest s, v-dipath uses at most $|N| - 1$ arcs. By Theorem 5.3, $d_v = y_v$ if no negative dicycle exists. If not, then a negative cycle must exist. □

5.2. Dijkstra's Algorithm

Dijkstra's Algorithm is a special case of Bellman-Ford Algorithm with the assumption that all arcs have non-negative costs. In this case $y = 0$ is feasible for the dual. Dijkstra's Algorithm starts with this feasible solution, and will uniformly raise y to find equality arcs that form a spanning tree rooted at s.

Lecture 6

Dijkstra's Algorithm and Applications of Shortest Dipaths

6.1. Dijkstra's Algorithm

Remark 6.1. After we raise potentials, what happens to the reduced costs? Consider uv.

1. If $u, v \notin N(T)$, then $\overline{w_{uv}}$ does not change.
2. If $u, v \in N(T)$, then $\overline{w_{uv}}$ does not change, also.
3. If $u \notin N(T)$ and $v \in N(T)$, then $\overline{w_{uv}}$ increases by t, still feasible.
4. If $u \in N(T)$ and $v \notin N(T)$, then $\overline{w_{uv}}$ decreases by t. To keep feasibility, we can maximize t by picking a smallest $\overline{w_{uv}}$ among $\delta(N(T))$. Such an arc becomes an equality arc; we will add it to the tree and continue from there.

Dijkstra's Algorithm

1. Initialization: Set $y_v = 0$ for all $v \in N$. Set T to have only s.
2. While T is not a spanning tree...
3. Pick $uv \in \delta(N(T))$ such that $\overline{w_{uv}} = \min\{\overline{w_e} : e \in \delta(N(T))\}$.
4. $y_w := y_w + \overline{w_{uv}}$ for all $w \in N \setminus N(T)$.
5. Add uv and v to T.

Justification

y is feasible at all times, including initialization by Remark 6.1. All arcs in T are equality arcs. Algorithm produces a spanning tree rooted at s. It must be a tree of shortest dipaths, because this gives an optimal solution to the Shortest Dipath Problem.

Remark 6.2. Dijkstra's Algorithm is faster than Bellman Ford's Algorithm. Once we add an arc and node to T, we do not change them again.

6.2. Applications of Shortest Dipaths

Network Reliability

Given network $D = (N, A)$, each arc e is assigned an associated reliability r_e where $0 < r_e \leq 1$, representing the probability that e is operational. For a dipath P, the reliability of P is $r(P) = \prod_{e \in P} r_e$. The goal here is to maximize the reliability among s, t-dipaths.

We will use the following transformation from products to additions:

$$\log r(P) = \sum_{e \in P} \log(r_e)$$

Also, we turn this into a minimization problem by multiplying -1 on all arcs, so now all arcs are positive. Then, apply Dijkstra's Algorithm.

Note that taking logs is time consuming, we can instead modify the Dijkstra's Algorithm to solve this directly without taking logs.

Lecture 7

Application to Shortest Paths, Maximum Flow, and Ford-Fulkerson Algorithm

7.1. Applications of Shortest Dipaths

Currency Exchange

Recall that Bellman-Ford detects negative dicycles.

Given a set of currencies, there is an exchange rate r_{uv} representing how much does 1 unit of currency u convert into currency v. The goal is to exchange a series of currencies back to the original one so that we make a profit.

Example 7.1. Suppose we have a sequences of changes:

$$< v_1, v_2, v_3, \ldots, v_k, v_1 >$$

we want

$$r_{v_1v_2} \cdot r_{v_2v_3} \cdots r_{v_kv_1} > 1$$

Note that

$$\log \prod_e r_e = \sum_e \log r_e$$

and

$$\log \prod_e r_e > 1 \iff \sum_e \log r_e > 0$$

When we maximize the sum, we minimize $-\log r_e$. Thus, we are looking for a negative directed cycle in this case. We can run Bellman-Ford algorithm to find such a negative directed cycle.

7.2. Maximum Flow

Given network $D = (N, A)$, two distinct nodes s, t (s is the source, t is the sink), arc capacities c. The goal is to maximize the amount of flow from s to t.

LP Formulation

Recall that s is the source node.

$$\begin{aligned}
&\max \quad x(\delta(s)) - x(\delta(\bar{s})) \qquad \text{(netflow out of } s\text{)} \\
&\text{s.t.} \\
&x(\delta(\bar{v})) - x(\delta(v)) = 0 \qquad (\forall v \in N \setminus \{s, t\}) \\
&x_e \leq c_e \qquad (\forall e \in A) \\
&x \geq 0
\end{aligned}$$

7.3. Ford-Fulkerson Algorithm

The idea is to repeatedly find s, t-paths to push flow on. Add flow to forward arcs, subtract flow on backward arcs.

To find such paths, we need to introduce the idea of residual digraphs.

Definition 7.2 (Residual Digraph)**.** Given D, capacities c, flow x, its corresponding residual digraph D' is defined by

1. $N(D') = N(D)$
2. For each arc $uv \in A$,
   a) if $c_{uv} > x_{uv}$, then add uv with residual $c_{uv} - x_{uv}$ (forward arcs), or
   b) if $x_{uv} > 0$, then add vu with residual x_{uv} (backward arcs).

Remark 7.3. If there is an s, t-dipath in D', then we can push flow in D.

Lecture 8

Ford-Fulkerson

1. Initialization: $x_e = 0$ for all $e \in A$. Form residual digraph D'.
2. While D' has an s,t-dipath...
3. Find s,t-dipath P in D'.
4. Push flow r along P ($+r$ for forward arcs, $-r$ for backward arcs), where r is the minimum residual in P.
5. Update D'.

Does it terminate?

- If c is integral, each iteration increases flow by at least 1. Thus, if the max flow has value k, then at most k iterations are needed. (This bound is tight in some extreme examples.)
- If c is rational, we can multiply c by the greatest common denominator and get a similar flow problem with integer capacities. It also terminates.
- If c is irrational, it is possible that the algorithm does not terminate.

A Note on Running Time

If the optimal flow is k, then there are examples of the network where the algorithm runs k iterations. If we always pick an augmenting path in D' with the fewest number of arcs, then we only need at most $|N| \cdot |A|$ iterations, even if the flow is irrational.

Does it terminate with a maximum flow?

The algorithm terminates when no s,t-dipaths exist in D'. Then there is an s,t-cut $\delta_{D'}(S)$ that is empty. For arcs out of $\delta_D(S)$, there are no forward version in D', so the flow is at capacity. For arcs into $\delta_D(S)$, there are no backward version in D', so the flow is 0. Then, there is no larger flow that is possible. For any s,t-cut, it limits the flow by its capacities of the outgoing arcs, i.e., if all outgoing arcs are full, it is a maximum flow.

Lecture 9

Max-Flow-Min-Cut Theorem and Combinatorial Applications

9.1. Max-Flow-Min-Cut Theorem

Remark 9.1. The total amount of flow equals the net flow at s.

Proposition 9.2. *Given $D = (N, A)$, nodes s, t, capacities c, the value of any s, t-flow is at most the capacity of any s, t-cut.*

Proof. For any flow x and any s, t-cut $\delta(S)$, the net flow of S equals the net flow of s, since the net flow at other nodes is 0 in Maximum Flow Problem. The value of x is

$$x(\delta(s)) - x(\delta(\bar{s})) = x(\delta(S)) - x(\delta(\overline{S})) \leq c(\delta(S))$$

The value of x is at most the capacity of $\delta(S)$. □

Theorem 9.3 (Max-Flow-Min-Cut Theorem)**.** *Given $D = (N, A)$, nodes s, t, capacities c, the maximum value of a flow equals the minimum capacity of an s, t-cut.*

Proof. Let x be a maximum flow. Then there is no s, t-dipath in the residual digraph D'; otherwise, Ford-Fulkerson can find a better flow. Then there exists an empty s, t-cut $\delta_{D'}(S)$ in D'. If $uv \in \delta_D(S)$, then uv is not an arc in D'. Then, $x_{uv} = c_{uv}$. If $uv \in \delta_D(\overline{S})$, then vu is not an arc in D'. Then, $x_{uv} = 0$. Therefore, the value of x is $x(\delta(S)) - x(\delta(\overline{S})) = c(\delta(S))$. By Proposition 9.2, x is a maximum flow, and $\delta(S)$ is a minimum capacity s, t-cut. □

9.2. Combinatorial Applications

Menger's Theorem

Given $D = (N, A)$, nodes s, t, if there are k arc-disjoint s, t-dipaths, then we need k arcs to remove to disconnect s, t. Menger's Theorem states that the maximum number of s, t-dipaths equals the minimum size of s, t-cut.

Theorem 9.4 (Arc-Disjoint Version Menger's Theorem)**.** *Given $D = (N, A)$ and nodes s, t, the maximum number of arc-disjoint s, t-dipaths equals the minimum size of an s, t-cut.*

We will use the following theorem to simply the arguments:

Theorem 9.5 (Flow Decomposition Theorem)**.** *Given $D = (N, A)$ with integral capacities c, if x is an integral s, t-flow of value k, then x is the sum of the characteristic vectors of k s, t-dipaths and any number of dicycles.*

Proof. We can find one s, t-dipath P. Then $x - x^P$ is an s, t-flow of value $k - 1$. Base case is a circulation with any number of dicycles. By induction, this completes the sketch. □

Lecture 10

Combinatorial Applications

10.1. Proving Menger's Theorem

Theorem 10.1 (Arc-Disjoint Version Menger's Theorem)**.** *Given $D = (N, A)$ and nodes s, t, the maximum number of arc-disjoint s, t-dipaths equals the minimum size of an s, t-cut.*

Proof. If there are k arc-disjoint s, t-dipaths, we must remove at least k arcs to disconnect s from t (at least one from each dipath). Then, any s, t-cut has size at least k.

We now consider D with capacity 1 on all arcs. By Theorem 9.3, there exists a flow x with the same value as the capacity of an s, t-cut $\delta(S)$, say this value is k. We may assume x is integral as c is integral. By Theorem 9.5, x is the sum of k s, t-dipaths and some dicycles. Since arc capacities are 1, each arc can be used in at most one dipath, so the k s, t-dipaths are arc-disjoint. Also, $\delta(S)$ has exactly k arcs. Hence, they are maximum and minimum. □

Node-Disjoint Version

Suppose we want to know how many nodes we need to remove to disconnect s from t. We assume arc st is not there.

If there are k node-disjoint s, t-dipaths, then we need to remove at least k nodes.

Theorem 10.2 (Node-Disjoint Version Menger's Theorem)**.** *If st is not an arc, then the maximum number of node-disjoint s, t-dipaths equals the minimum size of an s, t-separating set of nodes (a set of nodes whose removal disconnects s and t).*

Proof. To ensure each node is used by one s, t-dipath, we separate a node v into v^+ and v^- such that there is an arc from v^+ to v^- with capacity 1. The incoming arcs

to v become incoming arcs to v^+. The outgoing arcs to v become outgoing arcs to v^-.

Now, we set the capacity of any other nodes to ∞. Thus, a maximum flow is a node-disjoint s,t-dipath and a minimum cut can not use ∞ arcs so therefore the minimum cut must contain each arc from each node-disjoint s,t-dipath. This completes the proof. □

10.2. Matchings

Definition 10.3 (Matching)**.** A *matching* in a graph G is a set of edges, no two of which share a vertex.

Definition 10.4 (Cover)**.** A cover of a graph G is a set $C \subseteq V(G)$ such that every edge of G is incident to a vertex of C.

Remark 10.5. In general, the size of a matching is at most the size of a cover.

Theorem 10.6 (König's Theorem)**.** *In a bipartite graph, the size of a maximum matching equals the size of a minimum cover.*

Proof. Let $G = (X, Y)$ be a bipartite graph. Add a new node s that joins to all vertices in X and add a new node t that joins from all vertices in Y. We set capacity of the new arcs to 1 and set other capacities to ∞. A maximum flow corresponds to matching of the largest size. A minimum cut $\delta(S)$ must not use any ∞ arcs and $(A \setminus S) \cup (B \cap S)$. Arcs from $A \cap S$ to $B \setminus S$ do not exist in $\delta(S)$ (since their capacities are ∞ and the minimum cut does not contain any ∞ arcs), so our set is a cover with the same size as the matching. □

Lecture 11

Flows with Lower Bounds and Applications of Maximum Flow

11.1. Flows with Lower Bounds

Suppose now we introduce non-negative lower bounds $\ell \in \mathbb{R}^A$ for the arcs. A flow x must satisfy $\ell_e \leq x_e \leq c_e$ for all $e \in A$.

[Insert Example Here]

Without the constraint of lower bounds, there is an easy feasible solution to begin with, namely, the zero flow.

Once we obtain a feasible solution, the remainder is the same as before with a modification: when we form the backward arcs, we must consider the lower bounds of each arc.

Modification of Ford-Fulkerson

Given a feasible flow, we form residual digraph D' where for the backward arcs, we put residual $x_e - \ell_e$.

[Insert Example Here]

Generalized Max-Flow-Min-Cut Theorem

When there is no s,t-dipath in D', there is an empty s,t-cut $\delta_{D'}(S)$. If there is an arc $e \in \delta_D(S)$, since there is no arc $e \in \delta_{D'}(S)$, $x_e = c_e$. If there is an arc $e \in \delta_D(\overline{S})$, since $e \notin \delta_{D'}(S)$, $x_e = \ell_e$.

The net flow on $\delta(S)$ is

$$\ell(\delta(\overline{S})) - c(\delta(S))$$

This is an upper bound on any flow.

We will state the results without proof since we have discussed in Lecture 9.

Proposition 11.1. *For any flow x and any s,t-cut $\delta(S)$, the value of x is at most $c(\delta(S)) - \ell(\delta(\overline{S}))$.*

Theorem 11.2 (Generalized Max-Flow-Min-Cut Theorem)**.** *Given $D = (N, A)$ with c, ℓ, s, t, there is a maximum s,t-flow x whose value is the same as the minimum value of $c(\delta(S)) - \ell(\delta(\overline{S}))$ over all s,t-cuts $\delta(S)$.*

Note on Certificate of Infeasibility

We will not dive in too much detail about feasibility problem on lower bounded maximum flow problem. However, we will state the following theorem:

Theorem 11.3. *The network is infeasible if and only if there exists an s,t-cut $\delta(S)$ such that $c(\delta(S)) < \ell(\delta(\overline{S}))$.*

You can formulate the problem of finding a feasible flow into a minimum cost flow problem, which you will see in the assignment.

11.2. Applications of Maximum Flow

`[See slides]`

Lecture 12

Applications of Maximum Flow and Preflow-Push Algorithm

12.1. Applications of Maximum Flow

`[See slides]`

12.2. Preflow-Push Algorithm

A drawback of Ford-Fulkerson Algorithm is that every iteration requires an s,t-path. This is not ideal if s and t are far apart. We might push flow 1 from s to t every time. In contrast, preflow-Push Algorithm makes local adjustments to the flow.

Outline of Algorithm

1. Push all possible flow out of s.
2. While the flow is not feasible...
3. If we can push flow on uv, we do so. (Some requirements)
4. Otherwise, increase height on a node.

Main Differences Between Ford-Fulkerson and Pre-Flow Push

- Ford-Fulkerson: Maintain feasible flow; work towards optimality (no s,t-dipaths in the residual)

- Preflow-Push: Infeasible flow, maintain optimality condition; work towards feasibility.

Definition 12.1 (Preflow)**.** An s,t-preflow is a flow that satisfies $0 \leq x \leq c$ and $x(\delta(\overline{v})) \geq x(\delta(v))$ for all $v \in N \setminus \{s,t\}$. The excess at v is $e_x(v) = x(\delta(\overline{v})) - x(\delta(v))$.

Lecture 13

Preflow Push Algorithm

Definition 13.1 (Preflow)**.** An s,t-preflow is a flow that satisfies $0 \leq x \leq c$ and $x(\delta(\overline{v})) \geq x(\delta(v))$ for all $v \in N \setminus \{s,t\}$. The excess at v is $e_x(v) = x(\delta(\overline{v})) - x(\delta(v))$.

Note that we use $e(v)$ to represent the excess at v for a preflow.

In the residual digraph D', use r_e to represent the residual of each arc $e \in A(D')$. For each node, we define a height function $h(v)$.

Definition 13.2. A set of heights h for N is compatible with a preflow x if

- $h(s) = |N|$ *this remains true always*
- $h(t) = 0$
- $h(v) \geq h(u) - 1$ for all $uv \in A(D')$

This means that any arc in D' does not point "steeply down", although there is no restriction on how much you point up.

The algorithm will always maintain a preflow x and compatible heights h.

Preflow-Push Algorithm

1. Initialization:
   - Set preflow x with $x_e = c_e$ for all $e \in \delta_D(s)$, $x_e = 0$ otherwise.
   - Set $h(S) = |N|$, $h(v) = 0$ for all other nodes v.
2. While there exists $u \in N \setminus \{s, t\}$ where $e(u) > 0$ ...
3. If there exists $uv \in A(D')$ where $h(v) = h(u) - 1$. Then push $\min\{r_{uv}, e(u)\}$ on uv. *Push means add flow on forward arcs, subtract flow on backward arcs.*
4. Otherwise, increment $h(u)$ by 1. *This is called a relabel.*

Example 13.3. `See Slides`

Correctness of Preflow-Push

Lemma 13.4. *If preflow x and height h are compatible, then D' has no s, t-dipath.*

Proof. Suppose on the contrary that there exists an s, t-dipath $s = v_0, v_1, \ldots, v_k = t$ in D'. Note that $h(s) = |N|$ and $h(t) = 0$. Then $h(v_{i+1}) \geq h(v_i) - 1$ for each $i = 0, \ldots, k-1$. Adding all such inequalities together, we get $h(v_k) \geq h(v_0) - k$, so $0 \geq |N| - k$. Then $|N| \leq k$, which contradicts $|N| \geq k + 1$. □

Corollary 13.5. *If x is a feasible flow with compatible height h, then x is a maximum flow.*

Proof. Since x is feasible, then there is no s, t-dipath in D'. Then it is an optimal solution. □

We now prove that the algorithm maintains compatible preflow and heights.

Remark 13.6. If $e(u) > 0$, then there must be some arc in $\delta_{D'}(u)$, so we can always push flow out.

Lecture 14

Total Correctness of Preflow-Push Algorithm

14.1. Partial Correctness

Theorem 14.1. *The preflow-push algorithm maintains compatible preflow and heights.*

Proof. The initialization gives $h(s) = |N|$, $h(t) = 0$, $h(u) = h(v)$ for arc $uv \notin N$ where $u, v \notin S$. For arcs $sv \in A(D)$, only vs is in D' as $x_{sv} = c_{sv}$, so $h(s) = |N| \geq h(v) - 1 = -1$. Similar for $vs \in A(D)$. Heights are compatible at initialization.

Suppose we push flow on $uv \in A(D')$. We do not change heights, so compatibility remains for existing arcs in D'. The only possible new arc is vu in D', but $h(v) = h(u) - 1$ before the push, so $h(u) \geq h(v) - 1$, vu is compatible.

Suppose we relabel u. We want to relabel u since $e(u) > 0$ and for all arcs $uv \in A(D')$, $h(v) \geq h(u)$. By adding 1 to u, we get $h(v) \geq h(u) - 1$. Arcs of the form vu are still compatible, so the new heights are compatible. □

If the algorithm terminates, then there is no excess except s, t, so we have a feasible flow. Since there is no s, t-dipath exists in D' by Lemma 13.4, it must be a maximum flow.

14.2. Termination Proof

We will bound the number of push and relabel operations. If they are finite, then the algorithm terminates.

Bounding Number of Relabels

Bound the maximum possible value of $h(v)$, so the number of relabel operations is finite.

Lemma 14.2. *If $e(u) > 0$, then there exists a u, s-dipath in D'.*

Proof. We prove the contrapositive of the statement: if there is no u, s-dipath, then $e(u) = 0$.

Let T be the set of nodes u with no u, s-dipath in D'. We note that $\delta_{D'}(T) = \emptyset$, for otherwise we can find a dipath to s via the node not in T.

For arcs $vu \in A(\overline{T})$, either $vu \in A(D)$ and is empty, or $uv \in A(D)$ and is at capacity. The net flow of T is non-positive, but $e(u) \geq 0$ for all $u \in T$, so the net flow of T is non-negative. Thus, $e(u) = 0$ for all $u \in T$. □

Corollary 14.3. *$h(u) \leq 2|N| - 1$ for all $u \in N$ in the algorithm.*

Proof. Suppose the algorithm increases the $h(u)$ to $2|N|$ at some point. We want to relabel u, since $e(u) > 0$. By Lemma 14.2, there exists a u, s-dipath in D', which has length at most $|N| - 1$. By compatibility, height decreases by at most 1 in each arc, resulting in $h(s) > |N|$, contradiction. □

Corollary 14.4. *The total number of relabel operations is certainly bounded by $2|N|^2$.*

Proof. There are $|N|$ nodes, each with height at most $2|N|$, and the result follows. □

Lecture 15

Bounding Number of Push Operations

We divide push operations into 2 types: saturating pushes and non-saturating pushes.

Definition 15.1. (Saturating/Non-Saturating Push) A push on arc $uv \in A(D')$ is saturating if we push a flow of r_{uv}. It is non-saturating otherwise.

15.1. Saturating Pushes

Remark 15.2. If we do a saturating push on uv, then uv disappears from D' and vu will be in D'.

Theorem 15.3 (Bounding Saturating Pushes)**.** *The number of saturating pushes is at most* $2|N||A|$.

Proof. Given $uv \in A(D')$, how many saturating pushes are on uv at most?

Suppose we have a saturating push on uv in D', so $h(u) = h(v) + 1$. Then by Remark 15.2, uv disappears from D'. In order to push on uv again, we need to first push a flow on vu. In order to do so, we need to relabel v at least twice to obtain $h(v) = h(u) + 1$, so between 2 saturating pushes on uv, at least 2 relabel operations needs to occur. By Corollary 14.3, the maximum number of relabels on v is $2|N|$, so up to $|N|$ saturating pushes on uv can occur.

Since there are $2|A|$ possible arcs in D' (forward and backward), the number of saturating pushes is at most $2|A||N|$. □

15.2. Non-Saturating Pushes

For non-saturating pushes, we define a function

$$\Phi(x,h) = \Sigma(h(v) \mid v \in N, e(v) > 0)$$

This function is 0 at initialization, and it is always non-negative.

Theorem 15.4 (Bounding Non-Saturating Pushes)**.** *The number of non-saturating pushes is at most* $4|N|^2|A|$.

Proof. We first determine the effect on $\Phi(x,h)$ for each type of operation.

- A relabel operation is done on a node with positive excess, so $\Phi(x,h)$ is increased by 1. Over the algorithm, the maximum increase from relabelling is $2|N|^2$ by Corollary 14.4.
- Consider a saturating push on uv. This decreases $e(u)$ but increases $e(v)$. If $e(v) = 0$ before the push, then after the push, we add $h(v)$ to $\Phi(x,h)$. This adds at most $2|N| - 1$ to $\Phi(x,h)$, since $h(v) \leq 2|N| - 1$ by Corollary 14.3. The maximum increase from saturating pushes is $2|N||A|(2|N| - 1)$, since there are at most $2|N||A|$ saturating pushes by Theorem 15.3.
- Consider a non-saturating push on uv. Since the push flow is either e or r and non-saturating push does not push r, so it pushes e. Then $e(v)$ becomes positive, but $e(u)$ becomes 0. At most we add $h(v)$ to $\Phi(x,h)$ and subtract $h(u)$ from $\Phi(x,h)$, but since $h(u) = h(v) + 1$, $\Phi(x,h)$ decreases by at least 1. Since $\Phi(x,h) \geq 0$, the number of non-saturating pushes is at most $2|N|^2 + 2|N||A|(2|N| - 1) \leq 4|N|^2|A|$.

□

Summary

- Relabel: $2|N|^2$
- Saturating Push: $2|N||A|$
- Non-Saturating Push: $4|N|^2|A|$

Corollary 15.5. *The preflow-push algorithm terminates after at most* $8|N|^2|A| \sim 8|N|^4$ *operations.*

Lecture 16

Global Minimum Cuts and Hao-Orlin Algorithm

We want to find a minimum nontrivial cut in $D = (N, A)$ with capacity c.

[Example]

A practical example is the global network reliability problem.

Naive Algorithm

Pick all possible s, t pairs, run the Maximum Flow Algorithm on each. We need to run $|N|(|N| - 1)$ times.

Naive Simplification

The naive algorithm is a polynomial time algorithm, but we can do better: Pick a specific node s. It is on either side of a global minimum cut. We only need to find all s, t-cuts and t, s-cuts where s is fixed. This runs $2(|N| - 1)$ times. It turns out we can do better than this.

16.1. Hao-Orlin Algorithm

We will use Hao-Orlin algorithm to solve this problem more quickly, based on preflow-push.

Definition 16.1 (X, t-Cut)**.** An X, t-cut has the form $\delta(S)$ where $X \subseteq S$ and $t \notin S$.

Definition 16.2 (s-Cut)**.** An s-cut has the form $\delta(S)$ where $s \in S$ and nontrivial.

Generic Algorithm for Finding a Minimum s-Cut

Suppose we know how to solve the minimum X, t-cut problem.

1. $X = \{s\}$
2. While $X \neq N$,
3. Pick $t \notin X$, find a minimum X, t-cut
4. Add t to X
5. Output the minimum over all cuts found.

Theorem 16.3. *This algorithm finds a minimum s-cut.*

Proof. Suppose $\delta(S^*)$ is a minimum s-cut. We show that our algorithm either finds this one or the one with the same capacity.

Consider the first time we pick a random t such that t is not in S^*. At this point, $X \subseteq S^*$. Now $\delta(S^*)$ is an X, t-cut, which must have a minimum capacity. The algorithm must produce $\delta(S^*)$ or a cut of the same capacity. □

This finds a minimum s-cut in $|N| - 1$ iterations, but Hao-Orlin will solve this using the equivalent of preflow-pushes twice.

Preparation of Hao-Orlin Algorithm

Definition 16.4 (X-Preflow)**.** For $X \subseteq N$, an X-preflow is a flow where $e(v) \geq 0$ whenever $v \notin X$.

Moreover, the heights h are compatible with an X-preflow if

1. $h(v) = |N|$ for all $v \in X$
2. $h(t) \leq |X| - 1$
3. $h(v) \geq h(u) - 1$ for all $uv \in A(D')$

Definition 16.5 (Level)**.** A level k consists of all nodes with height k, denoted $H(k)$.

Definition 16.6 (Cut Level)**.** A cut level is a level k where no arc goes from $H(k)$ to $H(k-1)$ in D'.

The algorithm is based on the following results:

Lemma 16.7. *If $\delta(S)$ is an X,t-cut with $\delta_{D'}(S) = \emptyset$ and $e(v) = 0$ for all $v \in N \setminus (S \cup \{t\})$, then $\delta(S)$ is a minimum X,t-cut.*

Corollary 16.8. *If ℓ is a cut level and $e(v) = 0$ for all v with $h(v) < \ell$ except t, then $\{v : h(v) \geq \ell\}$ is a minimum X,t-cut.*

Lecture 17

Hao-Orlin Algorithm

Lemma 17.1. *If $\delta(S)$ is an X,t-cut with $\delta_{D'}(S) = \emptyset$ and $e(v) = 0$ for all $v \in N \setminus (S \cup \{t\})$, then $\delta(S)$ is a minimum X,t-cut.*

Proof. Take any X,t-cut $\delta(S)$. The net flow out of S is $x(\delta(S)) - x(\delta(\overline{S})) \leq c(\delta(S))$. They go to $N \setminus S$, so this net flow is equal to

$$\sum_{v \in N \setminus S} e(v) \geq e(t)$$

Thus,

$$e(t) \leq \sum_{v \in N \setminus S} e(v) = x(\delta(S)) - x(\delta(\overline{S})) \leq c(\delta(S))$$

Note that

$$e(t) = \sum_{v \in N \setminus S} e(v)$$

because $e(v) = 0$ for all $v \in N \setminus (S \cup \{t\})$. Also, we know

$$x(\delta(S)) - x(\delta(\overline{S})) = c(\delta(S))$$

because $\delta_{D'}(S) = \emptyset$, so out flow is at capacity and in flow is zero. Therefore,

$$e(t) = \sum_{v \in N \setminus S} e(v) = x(\delta(S)) - x(\delta(\overline{S})) = c(\delta(S))$$

The cut given in the hypothesis is minimum. □

Corollary 17.2. *If ℓ is a cut level and $e(v) = 0$ for all v with $h(v) < \ell$ except t, then $\{v : h(v) \geq \ell\}$ is a minimum X,t-cut.*

Hao-Orlin Algorithm

The algorithm runs preflow-push and maintains a cut level ℓ. The aim is to get rid of excess on nodes below ℓ. The algorithm keeps non-empty levels (except $|N|$) to have consecutive heights (t being the lowest).

1. Initialization: $X = \{s\}$, $t \in N \setminus X$, $h(S) = |N|$, $h(v) = 0$ otherwise, $\ell = |N| - 1$, send as much flow out of s as possible.
2. While $X \neq N$,
3. Run preflow-push with these exceptions:
   - We only push flow on v if $e(v) > 0$ and $h(v) < \ell$.
   - If we want to relabel v and v is the only node with height $h(v)$, then do not relabel. Instead, set $\ell = h(v)$.
   - If we want to relabel v to ℓ, then reset $\ell = |N| - 1$.
4. When no node satisfies $e(v) > 0$ and $h(v) < \ell$, store the cut $\{v : h(v) \geq \ell\}$. *This is a minimum X, t-cut.* Add t to X, send as much flow out of t as possible ($\delta(t) = \emptyset$ in D'). Set $h(t) = |N|$. Pick a new t from $N \setminus X$ with lowest height.
5. Pick a minimum cut among all stored cuts.

Lecture 18

Correctness of Hao-Orlin

Theorem 18.1. *The preflow and heights are compatible throughout the algorithm.*

Proof. Note that $h(v) = |N|$, for all $v \in X$ easily follows from the algorithm. Also, $h(v) \geq h(u) - 1$ holds by preflow-push.

Claim 18.2. The non-empty levels less than $|N|$ are consecutive.

Proof. Initially, there is only one node in level n, and all other nodes are at level 0. At later time, we do not relabel v if v is the only node with height $h(v)$, so this will keep levels consecutive. In the end of iteration, transitioning to a new iteration we move t level $|N|$, so the remaining non-X nodes still have consecutive levels. □

Claim 18.3. $h(t) \leq |X| - 1$.

Proof. Initially, $h(t) = 0 \leq |X| - 1 = 0$. In the general step when we move t to X, the next t (say t') has level $h(t)$ or $h(t) + 1$, since non-empty levels are consecutive. Originally, $h(t) \leq |X| - 1$. We add 1 to X after the move, and we might add 1 to $h(t)$, so this inequality holds after the move. □

Hence, the algorithm maintains compatible preflow and heights. □

Lemma 18.4. $h(v) \leq |N| - 2$ *for all* $v \notin X$.

Proof. We know that $h(t) \leq |X| - 1$ and there are $|N| - |X| - 1$ nodes not in $X \cup \{t\}$. Since non-empty levels are consecutive, the highest level outside X is at most $(|X| - 1) + (|N| - |X| - 1) = |N| - 2$. □

Theorem 18.5. *At all times,* ℓ *is always a cut level.*

Proof. Initially, $\ell = |N| - 1$. Level ℓ is empty by the above lemma, so it is a cut level. We change ℓ to something else only when we want to relabel v but it is the only node with its height. We want to relabel v because $e(v) > 0$ and no neighbour of v is one level below, so no arc in D' goes from level $h(v)$ to level $h(v) - 1$. Then $h(v)$ is a cut level, and we are correct in setting $\ell = h(v)$. □

Theorem 18.6. *The stored cuts in each iteration is a minimum X, t-cut.*

Hence, the algorithm produces the minimum s-cut.

Termination and Efficiency

Since $h(v) \leq |N| - 2$, so in total $|N|^2$ relabel operations. Also, saturating pushes and non-saturating pushes easily follow from preflow-push. The number of level setting operations is at most the number of relabel operations. Overall, this is roughly the same as running preflow-push case.

Lecture 19

Randomized Algorithm for Global Minimum Cuts

We consider the problem of global minimum cuts for undirected graphs $G = (V, E)$ and edge capacities $c \in \mathbb{R}^E$.

19.1. Random Contraction Algorithm

Definition 19.1 (Contraction)**.** Contraction on an edge uv is merging u, v into one vertex.

Remark 19.2. When we contract edges, we might obtain parallel edges.

Simplified Random Contraction Algorithm

1. Pick an edge at random and contract it.
2. Keep track of vertices that each contracted vertex represents.
3. Do this until we have 2 vertices left. Output the cut.

Remark 19.3. If we contract an edge in a minimum cut, then we can not produce such a cut. Generally, we have a good reason to assume that edges with small capacities are in a minimum cut. If we want to assign a probability of contraction to each edge, we might want to assign a smaller probability to these edges. Thus, we select an edge with probability that is proportional to its capacity.

Improved Random Contraction Algorithm

1. While $|V| > 2$,
2. Pick uv with probability $c_{uv} / \sum_{e \in E} c_e$
3. Contract uv

Theorem 19.4. *Let $\delta(S^*)$ be a global minimum cut. The probability that the algorithm produces $\delta(S^*)$ is at least*

$$1 / \binom{|V|}{2}$$

Proof. The given cut survives until the end if we never pick an edge in $\delta(S^*)$. We first calculate that the probability that we pick an edge in $\delta(S^*)$ in the first step. The numerator is $c(\delta(S^*))$ and the denominator is $\sum_{e \in E} c_e$. That is,

$$\frac{c(\delta(S^*))}{\sum_{e \in E} c_e}$$

Consider the cuts of the form $\delta(v)$ for all $v \in N$. Each edge uv appears in two such cuts, $\delta(u)$ and $\delta(v)$. Therefore,

$$\sum_{e \in E} c_e = \frac{1}{2} \sum_{v \in V} c(\delta(v)) \geq \frac{1}{2} \sum_{v \in V} c(\delta(S^*)) = \frac{1}{2}|V| c(\delta(S^*))$$

So, the probability that an edge from $\delta(S^*)$ is selected is

$$\frac{c(\delta(S^*))}{\sum_{e \in E} c_e} \leq \frac{c(\delta(S^*))}{\frac{1}{2}|V| c(\delta(S^*))} = \frac{2}{|V|}$$

The probability that the cut survives at the first contraction is $1 - \frac{2}{|V|}$. □